Boolean State Transformation

Boolean Gates

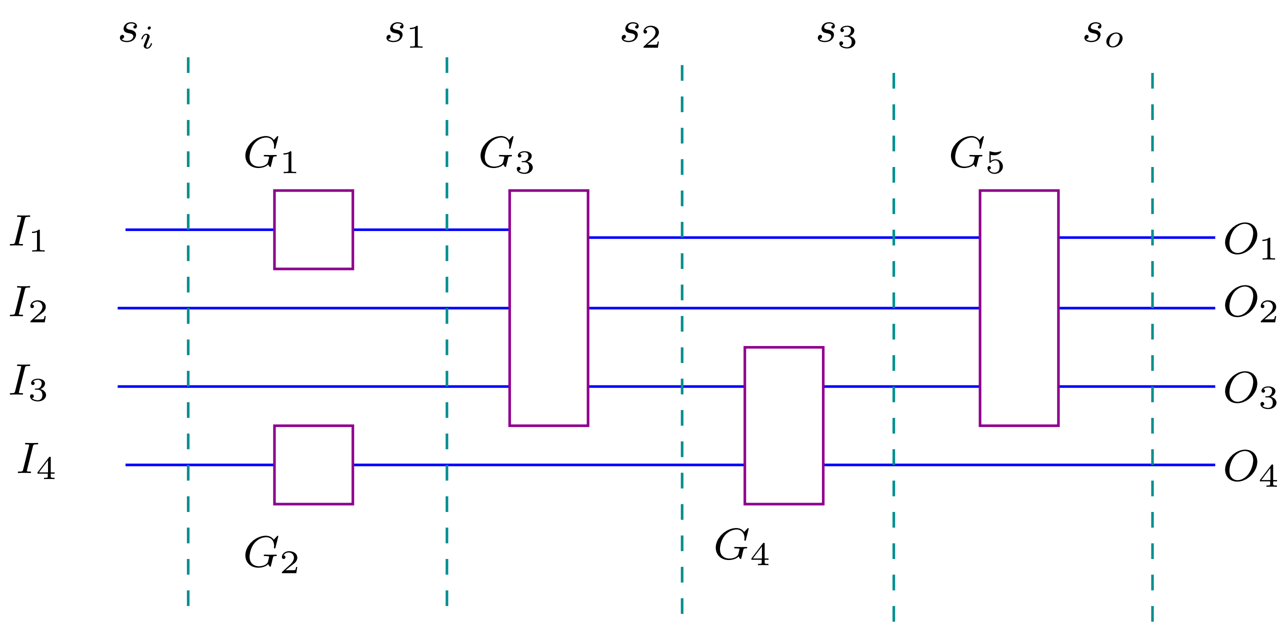
- Boolean gates transform the state of a Boolean system.
- Conventionally a Boolean gate is a map $f : \{0, 1\}^n \rightarrow \{0, 1\}$, where n is the number of input lines and there is only one output line.

  For a large n, an n-input gate may be decomposed in smaller realisable gates.

Boolean Gates

- But if the system state is represented by n-bits then a state transition map is $g : \{0, 1\}^n \rightarrow \{0, 1\}^n$.
- The map g may be viewed as an n input and n output gate. This also may be realised using smaller gates.
- Following diagram is an example.

Boolean Gate Array



Note

- s_i is the input state, s_o is the output state.
- s_1, s_2, s_3 are intermediate states.
- Transition from s_0 to s_1 is through the gates G_1, I, I, G_2, where I may be viewed as identity map.
- This transition may be viewed as a 4-bit transformation $G_1 \otimes I \otimes I \otimes G_2$.
- Other transitions are similar.

1-bit Boolean Gates

There are four one variable Boolean functions.

- Two constant functions: $c_0 : 0 \mapsto 0,\ 1 \mapsto 0$, $c_1 : 0 \mapsto 1,\ 1 \mapsto 1$,
- the identity map $i_1 : 0 \mapsto 0,\ 1 \mapsto 1$, and
- the not gate $\neg : 0 \mapsto 1,\ 1 \mapsto 0$.

1-bit Linear Algebra

If we encode Boolean **0** as $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and **1** as $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$, the following transformation matrices represent the four gates.

$$c_0 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, c_1 = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}, i_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \neg : \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

The i_1 and $\neg$ gate are invertible, but other two are not.

1-bit Linear Algebra

$$
\begin{aligned}
c_0(0) &= \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 0, \\
c_1(0) &= \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1, \\
\neg 0 &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1, \\
\neg 1 &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 0.
\end{aligned}
$$

1-Bit Boolean Transformation Matrix

- A 2×2 matrix over $\mathbb{F}_2$ is a valid transformation matrix for a single-bit if every column has exactly one 1.
- This restriction is due to our encoding of 0 and 1.
- We get $2 \times 2 = 4$ valid transformation matrices corresponding to c_0, c_1, i_1 and $\neg$.
- Only two of them are reversible.

Reversibility

- Reversibility of computation or invertibility of an operator is an issue even in classical computation.
- It is known from thermodynamics that there is no increase in entropy in a reversible process.
- But completely isentropic circuits are impossible to design.

Reversibility

- There are adiabatic circuits that use reversible logic to reduce power consumption during switching.
- Landauer's principle claims that any "logically irreversible" change in information causes more change in entropy.
- A physical reversibility demands logical reversibility.

Reversibility

- We shall see that state transition in a quantum mechanical system is reversible.
- Only the reversible classical logical gates may have quantum mechanical counterpart.
- So we explore the classical Boolean gates that are reversible as well as universal.

Tensor Product

We define the tensor product of two 2-dimensional vectors $\overrightarrow{x} = \begin{bmatrix} a \\ b \end{bmatrix}$ and $\overrightarrow{y} = \begin{bmatrix} c \\ d \end{bmatrix}$ over some field F as

$$\begin{bmatrix} a \\ b \end{bmatrix} \otimes \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} a \begin{bmatrix} c \\ d \end{bmatrix} \\ b \begin{bmatrix} c \\ d \end{bmatrix} \end{bmatrix} = \begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix}.$$

This can be generalised to higher dimensions.

2-Bit Boolean Data

1-bit Boolean data is encoded as $0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Two bit Boolean data may be viewed as the tensor product of two 1-bit vectors.

2-Bit Boolean Data

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ 0 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}.$$

2-Bit Boolean Data

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}.$$

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ 1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

2-bit Boolean Data

Four possible combinations of inputs,

00,01,10,11 are encode as four 4-vectors.

$$00 : \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, 01 : \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, 10 : \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, 11 : \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

This can be generalised for n-bits.

2-bit Boolean Gates

- There are sixteen 2-variable conventional Boolean functions e.g. and, nand, or, nor etc.
- They correspond to maps from $\{0,1\}^2 \rightarrow \{0,1\}$.
- But we are interested about state transition maps from $\{0,1\}^2 \rightarrow \{0,1\}^2$ that are reversible.
- There are total $4^4 = 256$ such maps.

2-Bit Boolean Transformation Matrix

- Similar to 1-bit transformations, a 2-bit transformation is a 4×4 matrix over $\mathbb{F}_2$. It is a valid if every column has exactly one 1.
- There are $4 \times 4 = 256$ valid transformations corresponding to 256 maps $\{0,1\}^2 \to \{0,1\}^2$.
- But only $4!$ among them are reversible, where every column has exactly one 1.

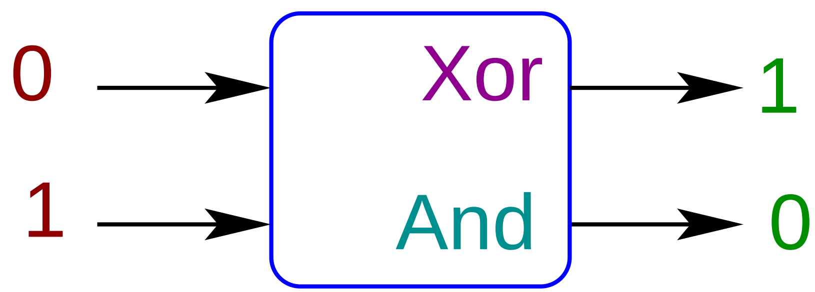
Universality and Reversibility

- But is there any universal gate out these 24 reversible gates?
- One possibility is to consider the tensor products of 1-bit reversible gates e.g. $\neg \otimes i_1$ or $\neg \otimes \neg$ - but they cannot be universal.
- Another temptation is to try with a pair of conventional gates and see what happens.

Xor-And Transformation

Here is an example of a 2-bit transformation matrix of a pair of conventional gates.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1(01) \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1(10) \\ 0 \end{bmatrix}$$



Note

- The transformation matrix of Xor-And pair is not invertible - the 4^{th}-row has all zeros.
- An invertible transformation matrix correspond to a permutation of rows of 4×4 identity matrix.

CNOT Gate

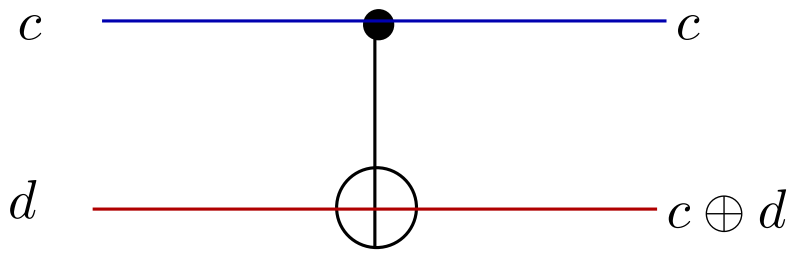
- A controlled not (CNOT) gate, has invertible transition matrix.
- Its inputs are (c, d), where c is the control input and d is the data input.
- The mapping is $(c, d) \mapsto (c, c \oplus d)$.

$$CNOT(c, d) = \begin{cases} (0, d) & \text{if } c = 0, \\ (1, \neg d) & \text{if } c = 1. \end{cases}$$

CNOT Transition Matrix

- $$CNOT(1,0) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = (1,1)$$
- CNOT is inverse of itself.
- CNOT(CNOT(c, d)) = CNOT$(c, c \oplus d)$ = $(c, c \oplus (c \oplus d)) = (c, d)$.

Diagram

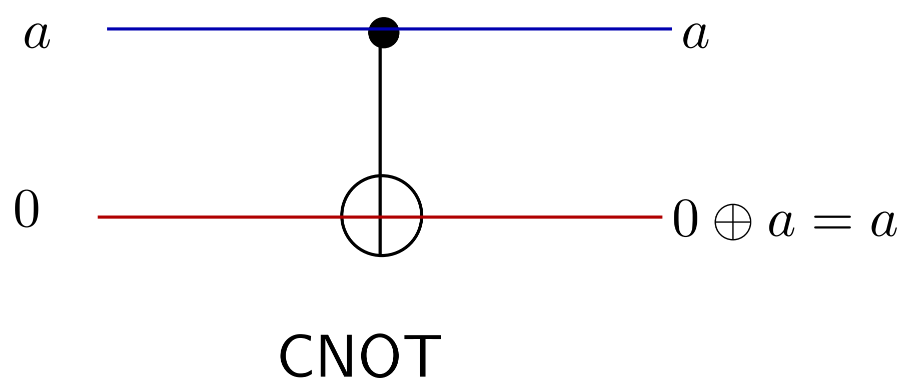


CNOT

CNOT can Copy a Bit

We can copy (clone) a bit using CNOT:

$(a, 0) \mapsto (a, a \oplus 0) = (a, a)$.



This is actually creating a FANOUT.

Note

- Tensor products of 1-bit reversible gate are $i_1 \otimes i_1$, $i_1 \otimes \neg$, $\neg \otimes i_1$, $\neg \otimes \neg$.
- CNOT cannot be realised using them.
- It is $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, where $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \end{bmatrix}$ and $\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \end{bmatrix}$.

2-bit Universal Gate is Impossible

The truth-table for a 2-bit reversible gate is as follows:

i_1	i_0	o_1	o_0
0	0	x_0	y_0
0	1	x_1	y_1
1	0	x_2	y_2
1	1	x_3	y_3

2-bit Universal Gate is Impossible

- The output of a reversible gate is a permutation of $\{00, 01, 10, 11\}$.
- It has exactly two 0's and two 1's per output column of the truth table.
- But $i_1 \wedge i_0$ has three 0's and $i_1 \vee i_0$ has three 1's in their output.
- None of them can be produced by a two bit reversible gate.

Reversible and Universal Boolean Gates

- The 2-bit CNOT gate is reversible.
- There are 24 reversible 2-Bit Boolean transformations. But that set cannot generate all logic function.
- 3-bit transformations are map from $\{0,1\}^3 \rightarrow \{0,1\}^3$. There are $8^8 = 16777216$ such transformations. But out of which $8! = 40320$ are reversible.

Universal Reversible Gates

- One 3-bit reversible transformation is Toffoli gate or CCNOT gate. It was invented by Tommaso Toffoli from Italy.
- The Toffoli gate or CCNOT gate is known to be universal.

Toffoli or CCNOT Gate

Following is the map of CCNOT gate: $(x, y, c) \mapsto (x, y, c \oplus (x \wedge y))$, where x, y remains unchanged,

$$c = \begin{cases} \neg c & \text{if } x = 1 = y, \\ c & \text{otherwise.} \end{cases}$$

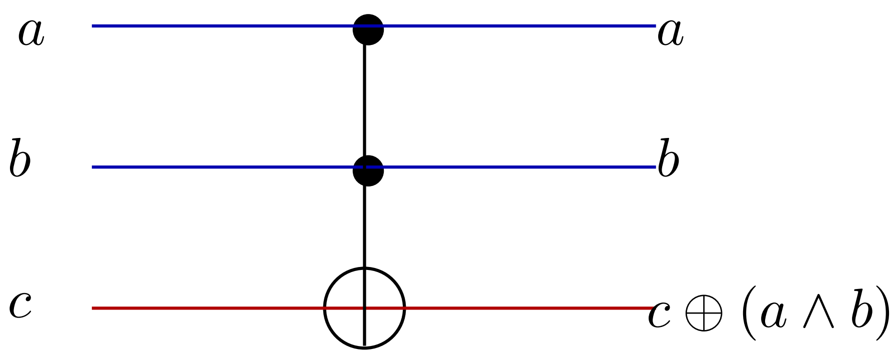
If $c = 0$, the 3^{rd} output is $0 \oplus (x \wedge y) = x \wedge y$.

CCNOT Gate Transition Matrix

CCNOT transition matrix is its own inverse.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Diagram



CCNOT

NOT, NAND and FANOUT from CCNOT

CCNOT gate can implement NOT, NAND and COPY/FANOUT operations if logic '0' and '1' are available.

$$(1, 1, c) \mapsto (1, 1, \neg c),$$

$$(a, b, 1) \mapsto (a, b, 1 \oplus (a \wedge b)) = (a, b, \neg(a \wedge b)),$$

$$(1, a, 0) \mapsto (1, a, 0 \oplus (1 \wedge a)) = (1, a, a).$$

OR using only CCNOT

We know that $\neg(\neg a \wedge \neg b) = a \vee b$.

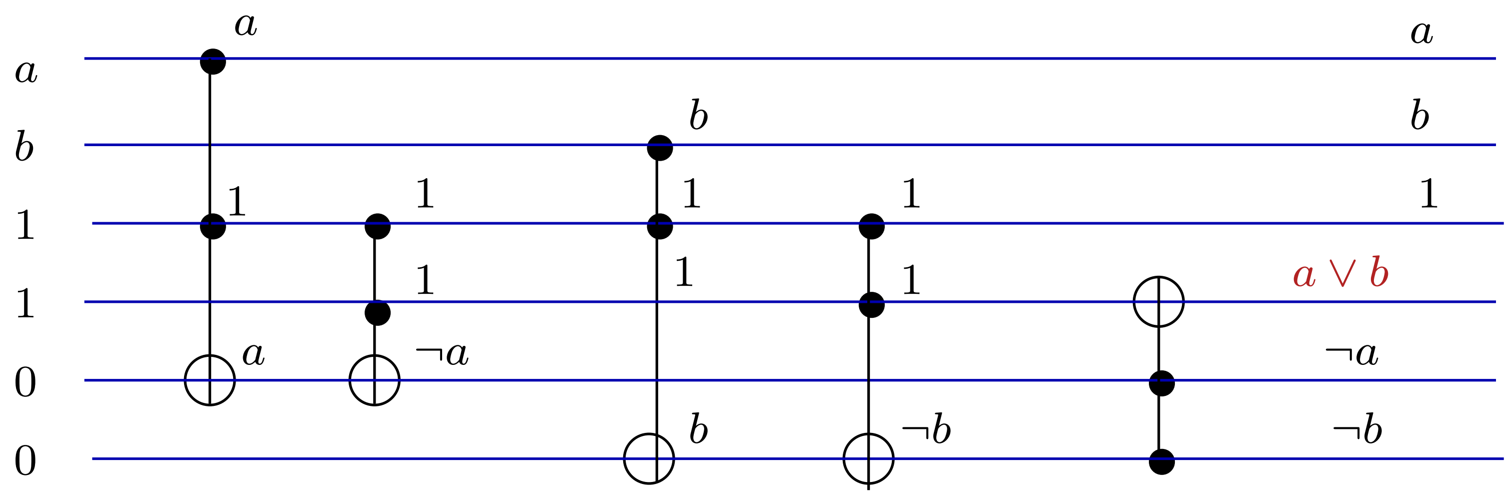
1. a and b are actual input. We use several 'ancilla' input.
2. Use two CCNOT gates to create a copies of a and b.
3. Use two CCNOT gates to compute $\neg a$ and $\neg b$.

OR using only CCNOT

4. Finally another CCNOT gate to compute $\neg(\neg a \wedge \neg b)$.

5. We are not taking care of crossovers.

OR using CCNOT



Note

Note that we have not taken care of cross-over of bits in the diagram. There are several ‘ancilla’ inputs in state ’0’ and ’1’. Also there are several useless outputs.

Note

It is not very difficult to create a 3-bit reversible transformation that will compute a 2-bit function e.g. or when the 3^{rd}-bit has a fixed value. Look at the following truth table.

Truth Table

a	b	c	o_1	o_2	o_3
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	0	1
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	1	0

When $c = 0$, $o_3 = a \vee b$. Other bits are filled to maintain reversibility.

Transition Matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

There are three permutations $(2, 3), (4, 5), (6, 7)$.

OR using CCNOT and NOT

- Another way of getting OR using CCNOT and NOT are as follows.
- We negate the input a, b and keep c unchanged.
- This can be done by applying the transformation

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

to a 3-bit vector.

Tensor Product of Matrices

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \otimes \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} & a_{12} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \\ a_{21} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} & a_{22} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22}. \end{bmatrix}$$

Tensor Product $\neg \otimes \neg \otimes I$

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Tensor Product $\neg \otimes \neg \otimes I$

We encode $(x, y, c) : (0,0,0), \cdots, (1,1,1)$ as an 8-dimensional Boolean vectors $000 \equiv (1, 0, \cdots, 0), \cdots, 111 \equiv (0, \cdots, 0, 1)$.

Operator $\neg \otimes \neg \otimes I$ gives the desired result.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} : (1,0,0) \mapsto (0,1,0).$$

Tensor Product $CCNOT \circ (\neg \otimes \neg \otimes I)$

- The transformation $(\neg \otimes \neg \otimes I)$ is clearly reversible (universal property).
- $CCNOT \circ (\neg \otimes \neg \otimes I)$ transforms $(x, y, c) \mapsto (\neg x, \neg y, c \oplus (\neg x \wedge \neg y))$.
- We can apply $\neg \otimes \neg \otimes \neg$ on the result. This gives us $x \vee y$ when $c = 0$.

Tensor Product ($I \otimes I \otimes \neg$)

$$(\neg \otimes \neg \otimes \neg) \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$(\neg \otimes \neg \otimes \neg) : (0, 1, 0) \mapsto (1, 0, 1)$$

Fredkin Gate

- Another important 3-input reversible gate is the Fredkin gate.
- The input-output relation is $(x, y, c) \mapsto (cy + \overline{c}x, cx + \overline{c}y, c)$ i.e. $(x, y, 0) \mapsto (x, y, 0)$ and $(x, y, 1) \mapsto (y, x, 1)$.
- If $c = 0$, x, y remains unchanged. If $c = 1$, the outputs are interchanged.

Fredkin Gate is Universal

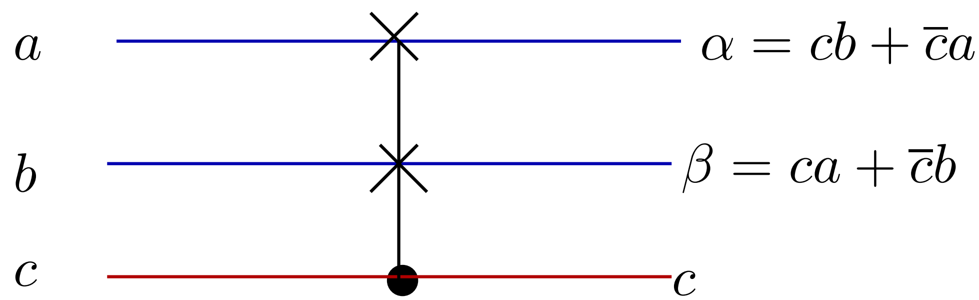
- CROSSOVER: $(a, b, 1) \mapsto (b, a, 1)$,
- NOT and FANOUT: $(1, 0, a) \mapsto (\overline{a}, a, a)$,
- AND: $(0, b, a) \mapsto (a \wedge b, \overline{a} \wedge b, a)$,

Fredkin Gate Transition Matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Clearly Fredkin gate is its own inverse.

Diagram

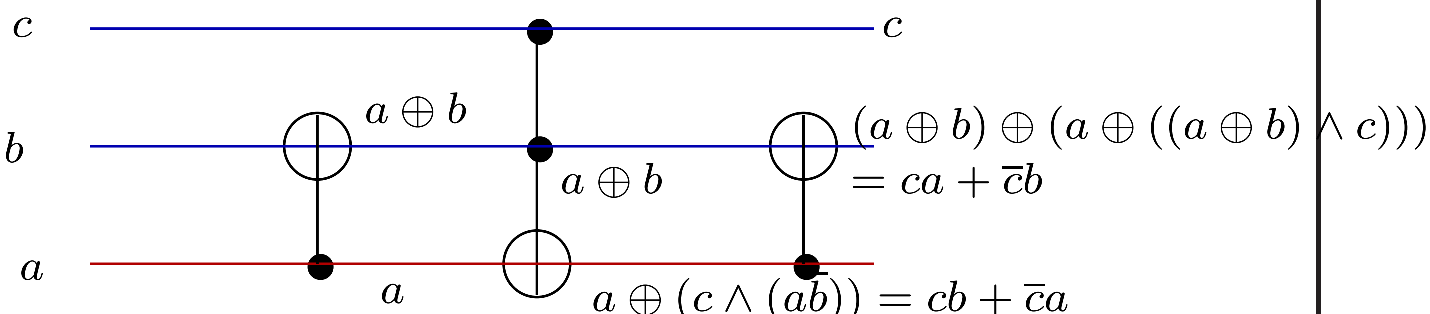


Fredkin

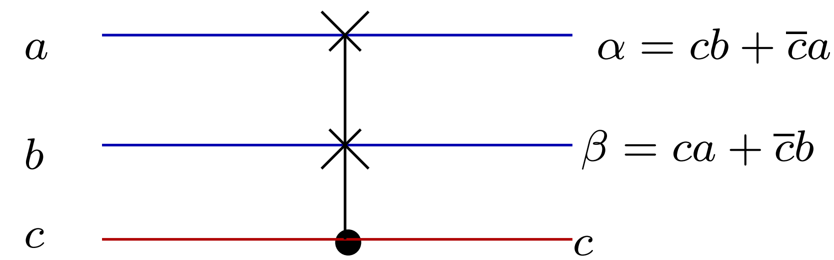
Note

Both in case of Toffoli gate and in Fredkin gate we need some ‘ancilla’ input bits in state ‘0’ or ‘1’ to make them universal. They also produce useless outputs not used in subsequent stages.

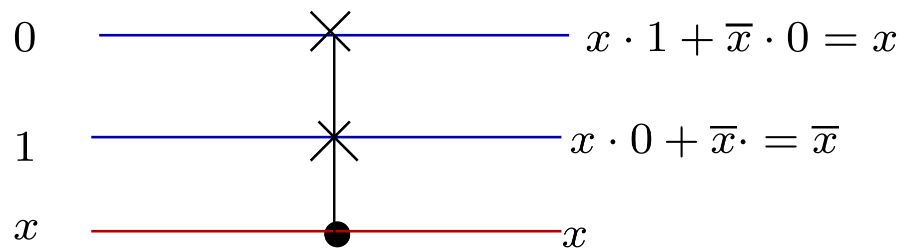
Fredkin using CNOT & CCNOT



NOT and FANOUT

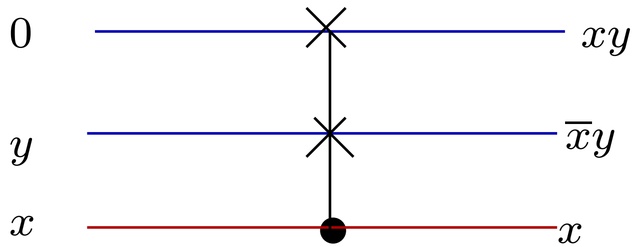


Fredkin

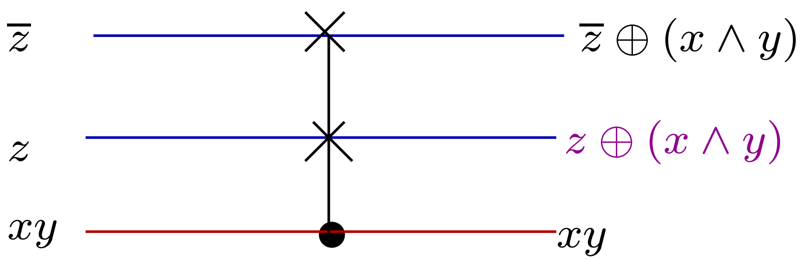


NOT and FANOUT

AND and CCNOT



AND



CCNOT

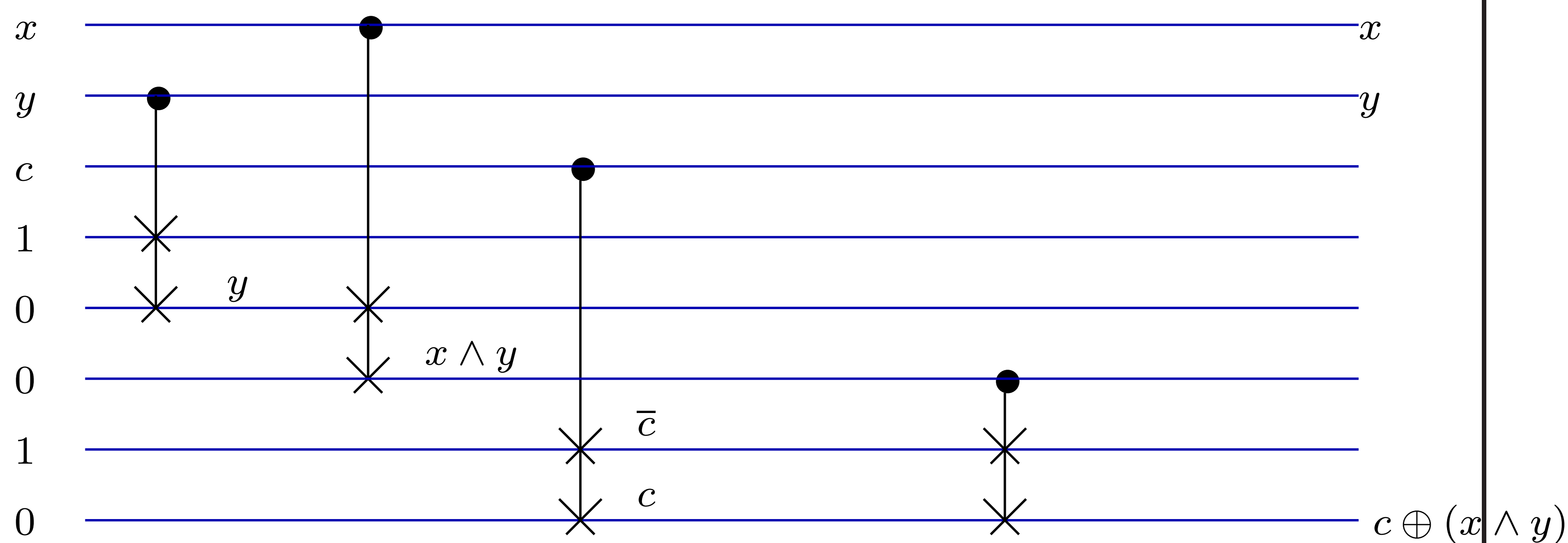
CCNOT using Fredkin

1. c, x and y are actual input. We use several 'ancilla' input.
2. Use a Fredkin gate to create a copy of y.
3. Use the second Fredkin gate to compute $x \wedge y$.

CCNOT using Fredkin

4. Use the third Fredkin gate to copy c.
5. Finally the fourth Fredkin gate computes $c \oplus (x \wedge y)$.
6. There are several output containing useless data.

CCNOT using Fredkin



Note

- We have several reversible and universal classical gates.
- Their quantum mechanical counterpart can be used to simulate classical computation.

Probabilistic Circuit

- The linear algebra formalism of classical bits and gates can be generalised to probabilistic circuits.
- Suppose a single-bit is at state 0 with probability p and at state 1 with a probability $1 - p$.
- It is represented as a 2-dimensional real vector $\begin{bmatrix} p \\ 1-p \end{bmatrix}$, where $p \in [0, 1]$.

1-bit Probabilistic Circuit

Applying our old transformation matrices we get,

$$c_0 \begin{bmatrix} p \\ 1-p \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p \\ 1-p \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

$$\neg \begin{bmatrix} p \\ 1-p \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} p \\ 1-p \end{bmatrix} = \begin{bmatrix} 1-p \\ p \end{bmatrix}$$

Transformation matrix for probabilistic bits is left stochastic (each column adds to 1).

2-bit Probabilistic Circuit

If we consider two bits so that

- the probability is p_0 for the 0^{th}-bit to be 0 and the probability is p_1 for the 1^{st}-bit to be 0.
- The probabilities of 00, 01, 10, 11 are $p_1 p_0, p_1(1-p_0), (1-p_1)q_0, (1-p_1)(1-p_0)$ respectively.

2-bit Probabilistic Circuit

The joint probabilities of two bits may be represented as the following tensor product -

$$\begin{bmatrix} p_1 \\ 1 - p_1 \end{bmatrix} \otimes \begin{bmatrix} p_0 \\ 1 - p_0 \end{bmatrix} = \begin{bmatrix} p_1 p_0 \\ p_1(1 - p_0) \\ (1 - p_1)p_0 \\ (1 - p_1)(1 - p_0) \end{bmatrix}.$$

Xor-And on Probabilistic Bits

If we apply Xor-And transformation on two probabilistic bits, we get

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 p_0 \\ p_1(1-p_0) \\ (1-p_1)p_0 \\ (1-p_1)(1-p_0) \end{bmatrix} = \begin{bmatrix} p_1 p_0 \\ (1-p_1)(1-p_0) \\ p_1(1-p_0) + (1-p_1)p_0 \\ 0 \end{bmatrix}.$$

The interpretation of transformation makes sense.

CNOT on Probabilistic Bits

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_1 p_0 \\ p_1(1-p_0) \\ (1-p_1)p_0 \\ (1-p_1)(1-p_0) \end{bmatrix} = \begin{bmatrix} p_1 p_0 \\ p_1(1-p_0) \\ (1-p_1)(1-p_0) \\ (1-p_1)p_0 \end{bmatrix}.$$

Again it has meaningful interpretation.

References

[ERWP] Quantum Computing: A Gentle Introduction by Eleanor Rieffel & Wolfgang Polak, Pub. MIT Press, 2011, ISBN 978-0-262-52667-8.

[MNIC] Quantum Computation and Quantum Information by Michael A Nielsen & Isaac L Chuang, Pub. Cambridge University Press, 2002, ISBN 81-7596-092-2.

[SA] Quantum Computing Since Democritus by Scott Aaronson, Pub. Cambridge University Press, 2013, ISBN 978-0-521-19956-8.

[AAM] Classical and Quantum Computation by A Yu Kitaev, A H Shen & M N Vyalyi, Pub. American Mathematical Society (GSM vol 47) 2002, ISBN 978-1-4704-0927-2.

References

[PRM] An Introduction to Quantum Computing by Phillip Kaye, Raymond Laflamme & Michele Mosca, Pub. Oxford University Press, 2007, ISBN 978-0-19-923677-0.